

# Secure Resource Sharing for Embedded Protected Module Architectures

Jo Van Bulck, Job Noorman, Jan Tobias Mühlberg and Frank Piessens

August 24, 2015



# Contents

1. Embedded Problem Domain
2. Protected Module Architectures
3. Motivation
4. Logical File Access Control
5. Conclusion

*“Embedded-systems security is,  
for lack of a better word, a mess.”*

– John Viega & Hugh Thompson

VIEGA John, THOMPSON Hugh, *The state of embedded-device security (spoiler alert: It's bad)*, IEEE Security & Privacy (10.5), September 2012, pp. 68-70.

# Software Isolation

## Conventional

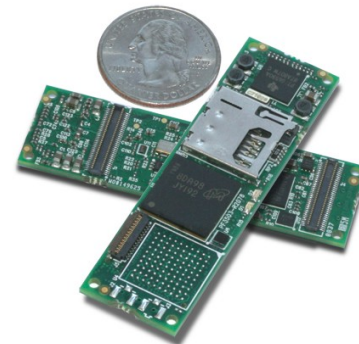
- Relatively expensive
- Power-consuming

=> Virtual memory & kernel mode

## Embedded

- Cheap
- Low power

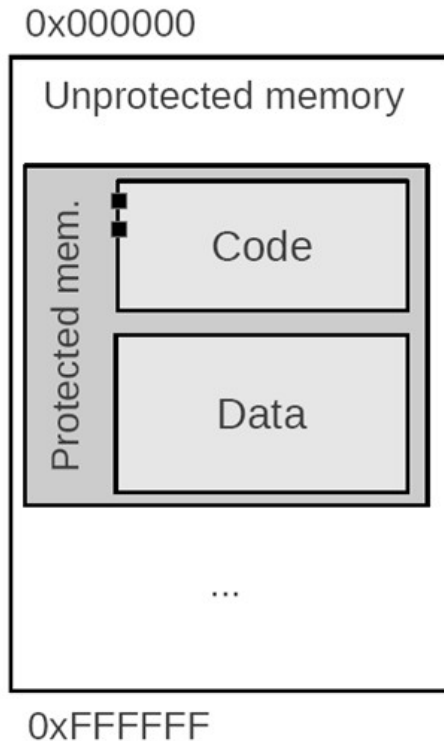
=> *Single-address-space*



# Contents

1. Embedded Problem Domain
2. Protected Module Architectures
3. Motivation
4. Logical File Access Control
5. Conclusion

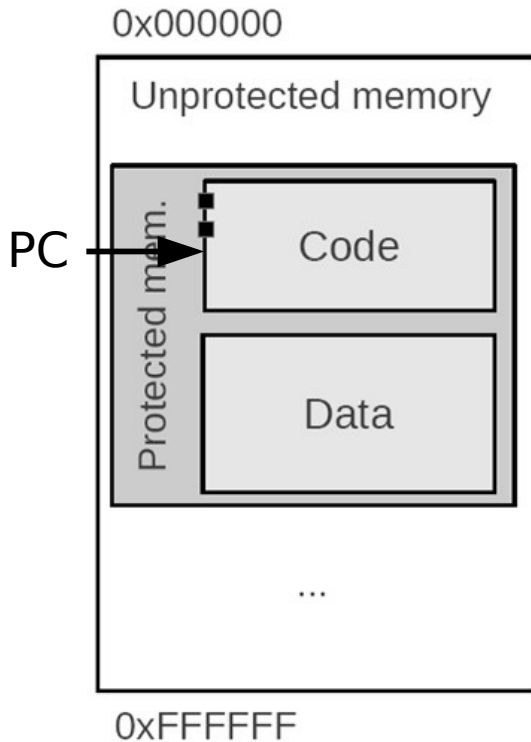
# Protected Module Architectures



- **Isolated execution** areas in a single-address-space
- **Program counter** based access control mechanism

From \ to	Protected			Unprotected
	Entry	Code	Data	
Protected	r-x	r-x	rw-	rwX
Unprotected / other SPM	r-x	r--	---	rwX

# Protected Module Architectures



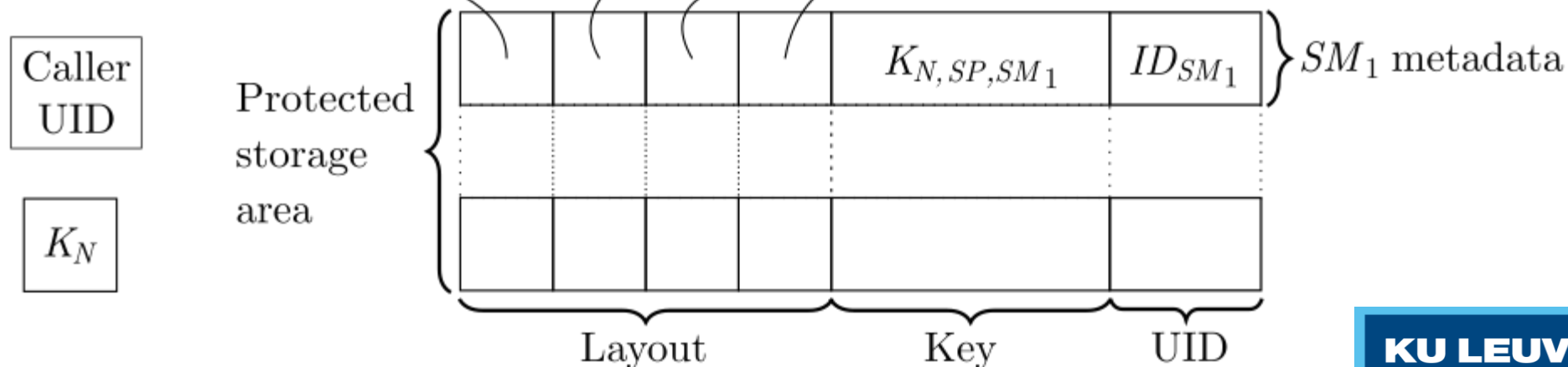
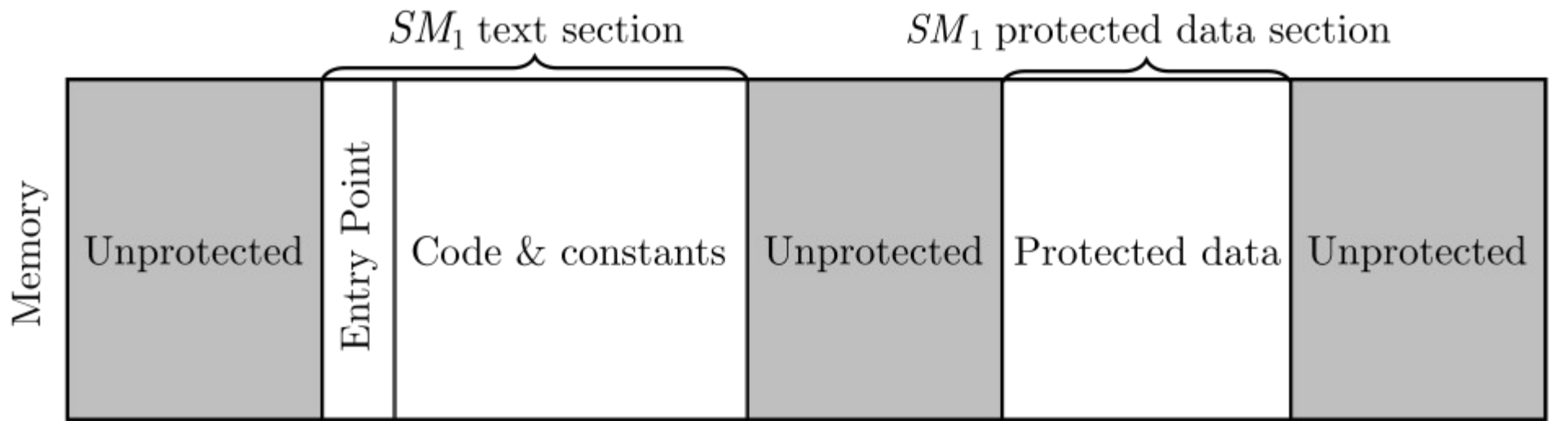
- **Isolated execution** areas in a single-address-space
- **Program counter** based access control mechanism

From \ to	Protected			Unprotected
	Entry	Code	Data	
Protected	r-x	r-x	rw-	rwX
Unprotected / other SPM	r-x	r--	---	rwX

# Sancus

- **Hardware-level PMA**
- **Zero-software TCB**
  - strong attacker model
- SM == unit of **protection / authentication**
  - hardware UID and cryptographic key per SM
  - `sancus_verify_address` & `sancus_get_caller_id`

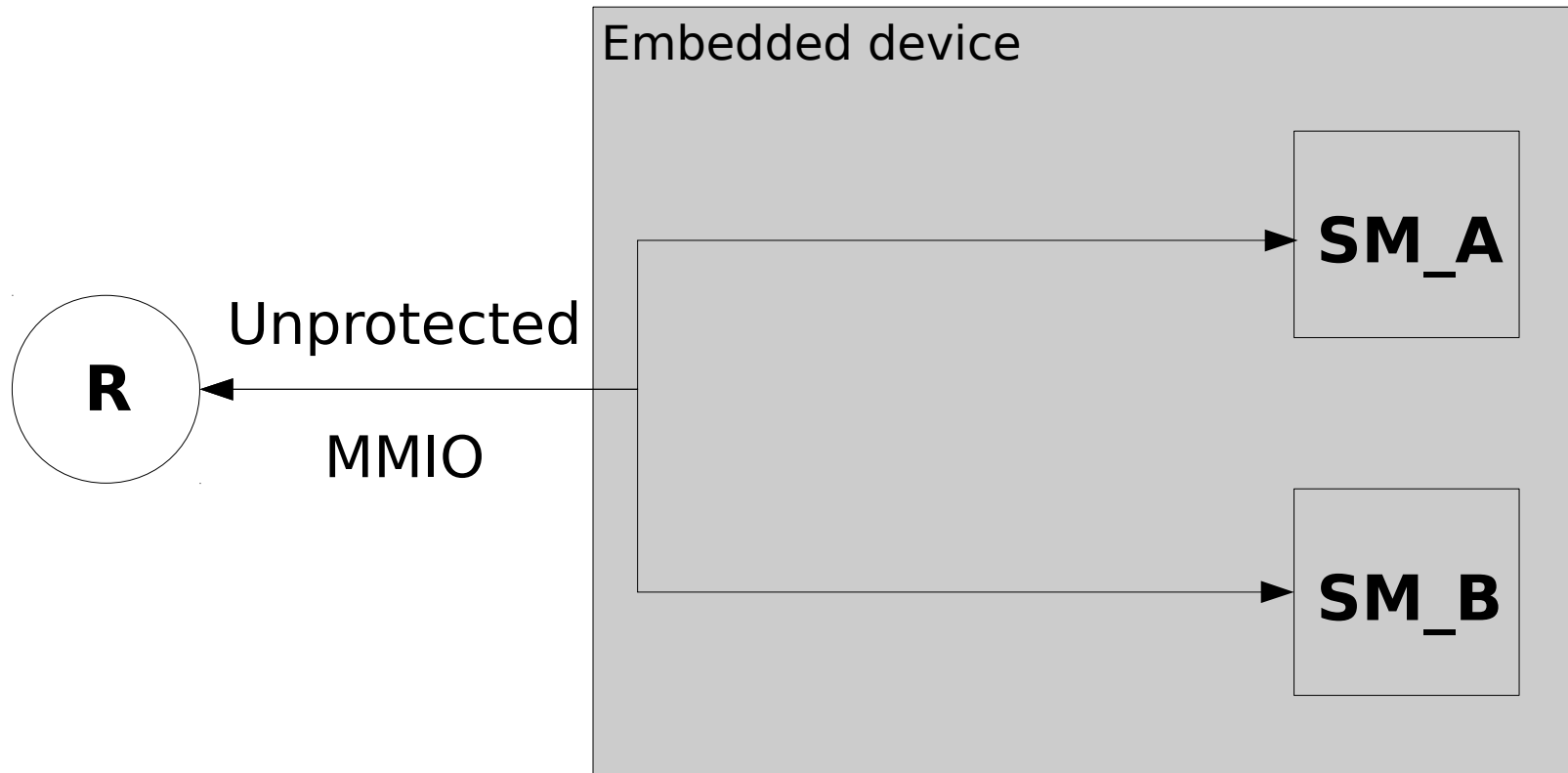




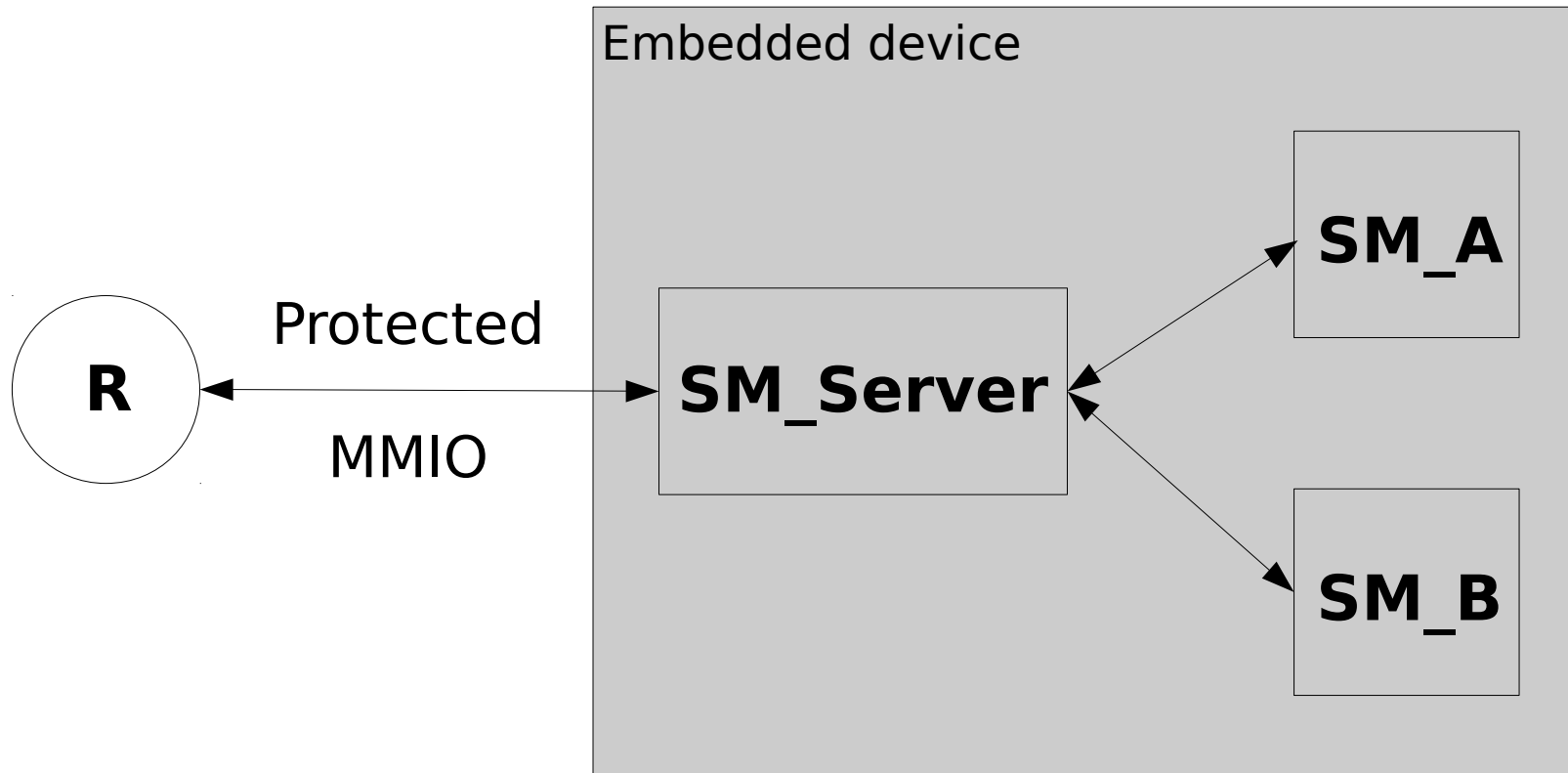
# Contents

1. Embedded Problem Domain
2. Protected Module Architectures
3. Motivation
4. Logical File Access Control
5. Conclusion

# Resource Sharing Approach



# Resource Sharing Approach



# Secure Resource Sharing

Sancus secludes SMs in protection domains:

- ☺ hardware-enforced **security guarantees**
- ☹ no **secure sharing** of platform resources

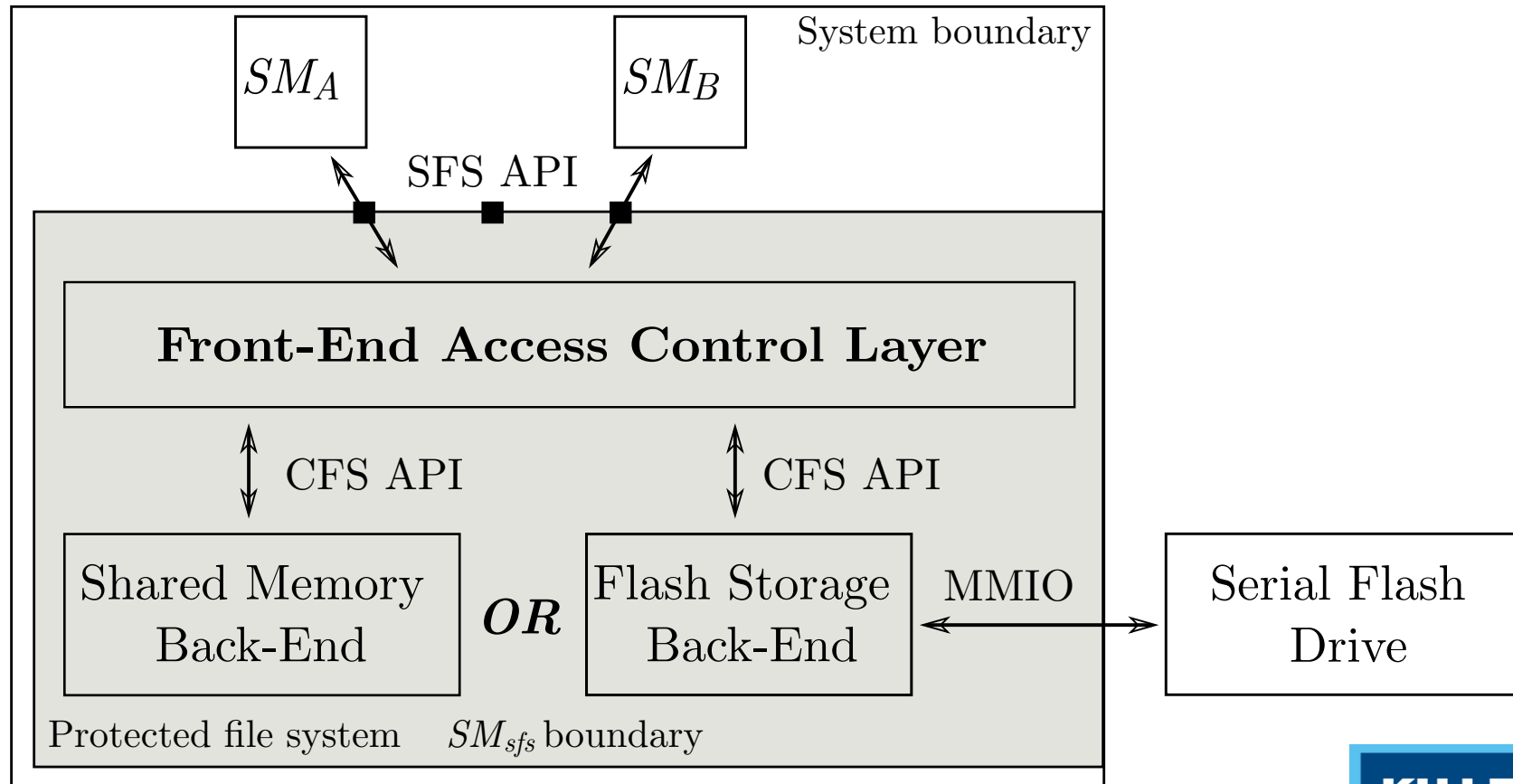
=> *protected “OS” modules* to supplement hw

- <> monolithic privileged kernel
- ~ extreme microkernel idea

# Contents

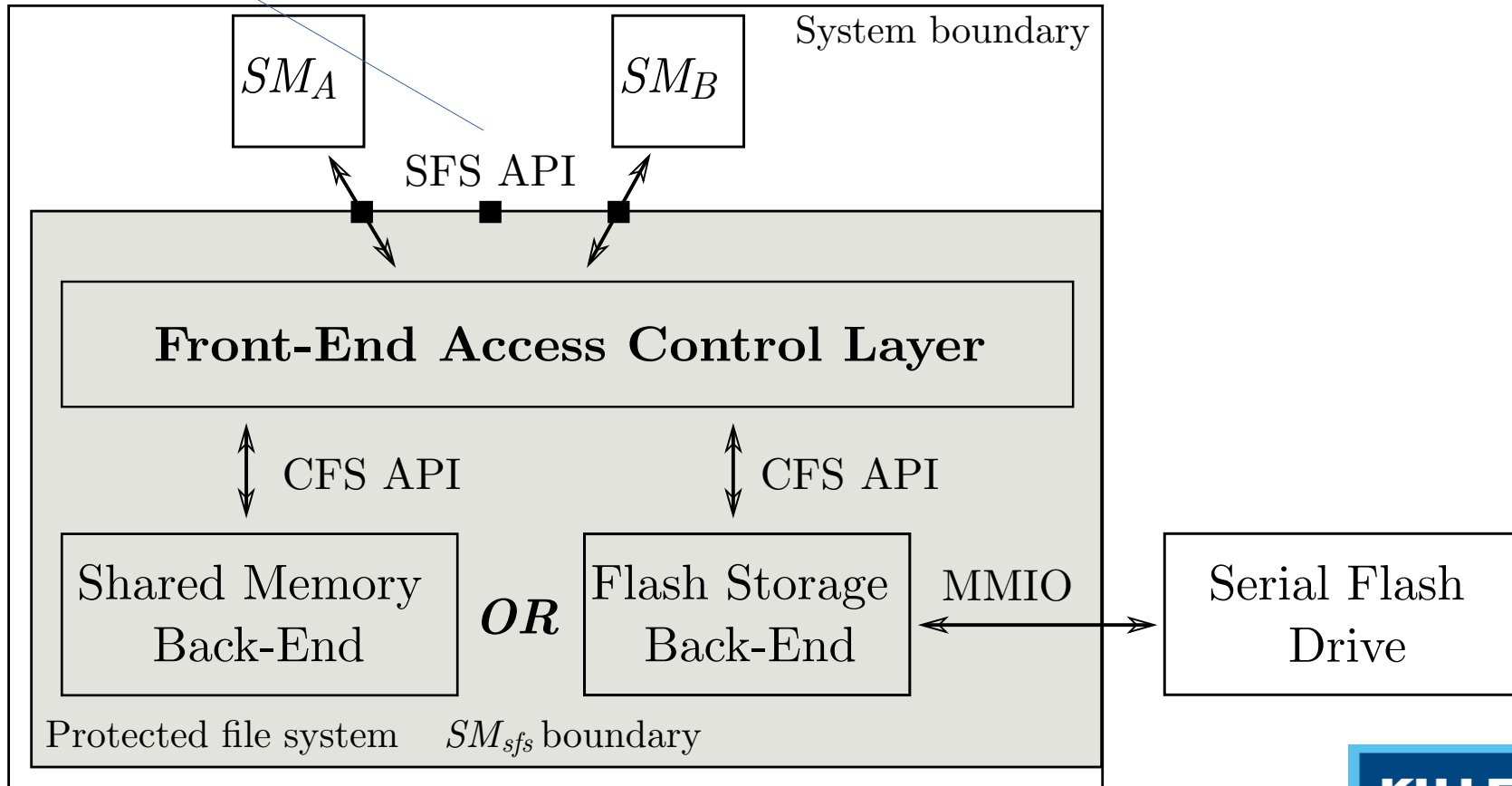
1. Embedded Problem Domain
2. Protected Module Architectures
3. Motivation
4. Logical File Access Control
5. Conclusion

# Sancus File System (SFS)



UNIX like file system API (incl. chmod)

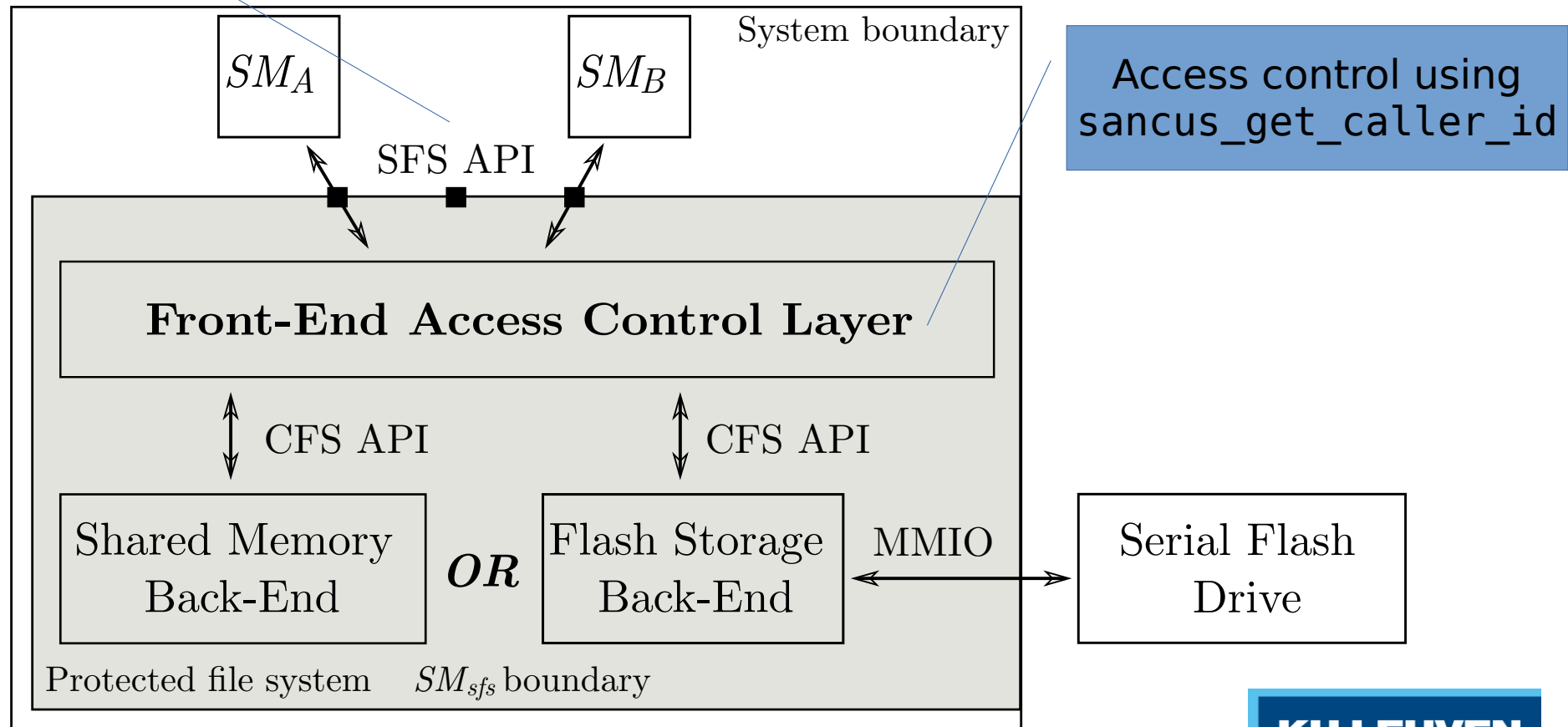
# Sancus File System (SFS)





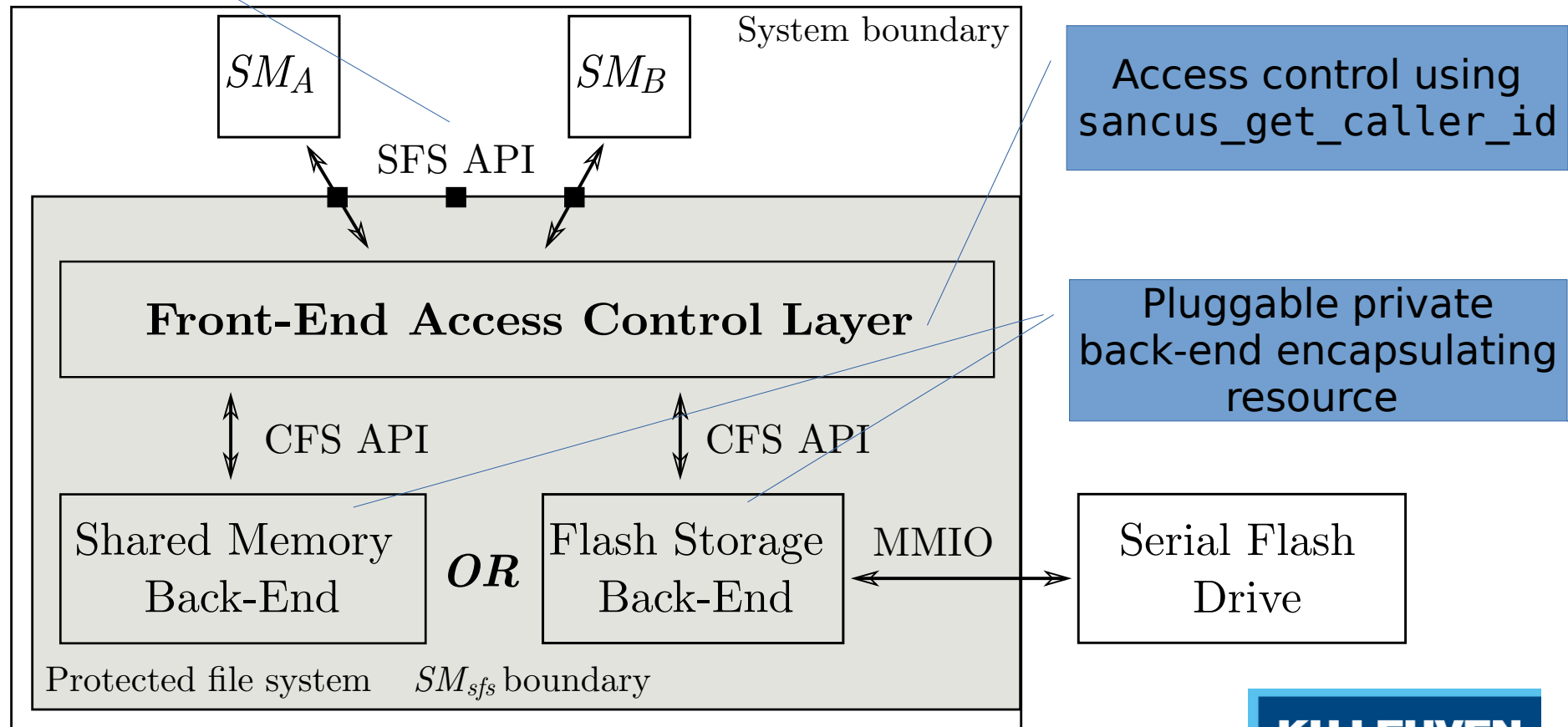
UNIX like file system API (incl. chmod)

# Sancus File System (SFS)



UNIX like file system API (incl. chmod)

# Sancus File System (SFS)



```
[clientA] revoking B permissions
[sfs-ram] INFO::sfs_chmod: trying to modify ACL for file 'a'
[sfs-ram] WARNING::ACL entry currently open; setting to SFS_NIL
[sfs-ram] INFO::sfs_chmod: trying to modify ACL for file 'b'
[sfs-ram] WARNING::ACL entry currently open; setting to SFS_NIL
[sfs-ram] INFO::sfs_dump: dumping global protected ACL data structures:
-----
FILE with name 'b' at 0x554; open_count = 2; next_ptr = 0x54c
    PERM (2, 0xff) at 0x586; file_ptr = 0x554; next_ptr = 0x58e
    PERM (3, 0x00) at 0x58e; file_ptr = 0x554; next_ptr = 0
FILE with name 'a' at 0x54c; open_count = 2; next_ptr = 0
    PERM (2, 0xff) at 0x576; file_ptr = 0x54c; next_ptr = 0x57e
    PERM (3, 0x00) at 0x57e; file_ptr = 0x54c; next_ptr = 0
-----
[sfs-ram] INFO::sfs_dump: dumping global protected file descriptor cache:
(0, 0x576); (1, 0x586); (2, 0x57e); (3, 0x58e); (4, 0x0); (5, 0x0); (6, 0x0); (7, 0x0);
[clientA] accessing B files (shouldn't work)
[clientB] accessing bunch of files
[sfs-ram] INFO::sfs_getc: read a char from file with fd 2
[sfs-ram] ERROR::permission check failed.
[sfs-ram] INFO::sfs_getc: read a char from file with fd 3
[sfs-ram] ERROR::permission check failed.
[sfs-ram] INFO::sfs_putc: write a char to file with fd 3
[sfs-ram] ERROR::permission check failed.
[clientA] closing b files
```

# Access Control Overhead

Majority of cycles caused by **SM switching**

Relative access control overhead decreases with the amount of work done in the back-end

☹ Protected shared memory back-end

☺ Flash Coffee FS: 20% for getc and 15% for putc

# Contents

1. Embedded Problem Domain
2. Protected Module Architectures
3. Motivation
4. Logical File Access Control
5. Conclusion

# Conclusion

- Generic **resource sharing** mechanism
- Confined and explicit **TCB**:
  - attestable via `sancus_verify`
  - principle of least privilege
- **Supplement** hw-enforced security guarantees
  - build upon hw primitives (isolation + caller auth)
  - sw-based access control guarantees

# Secure Resource Sharing for Embedded Protected Module Architectures

Jo Van Bulck, Job Noorman, Jan Tobias Mühlberg and Frank Piessens

<https://distrinet.cs.kuleuven.be/software/sancus/wistp2015/>

